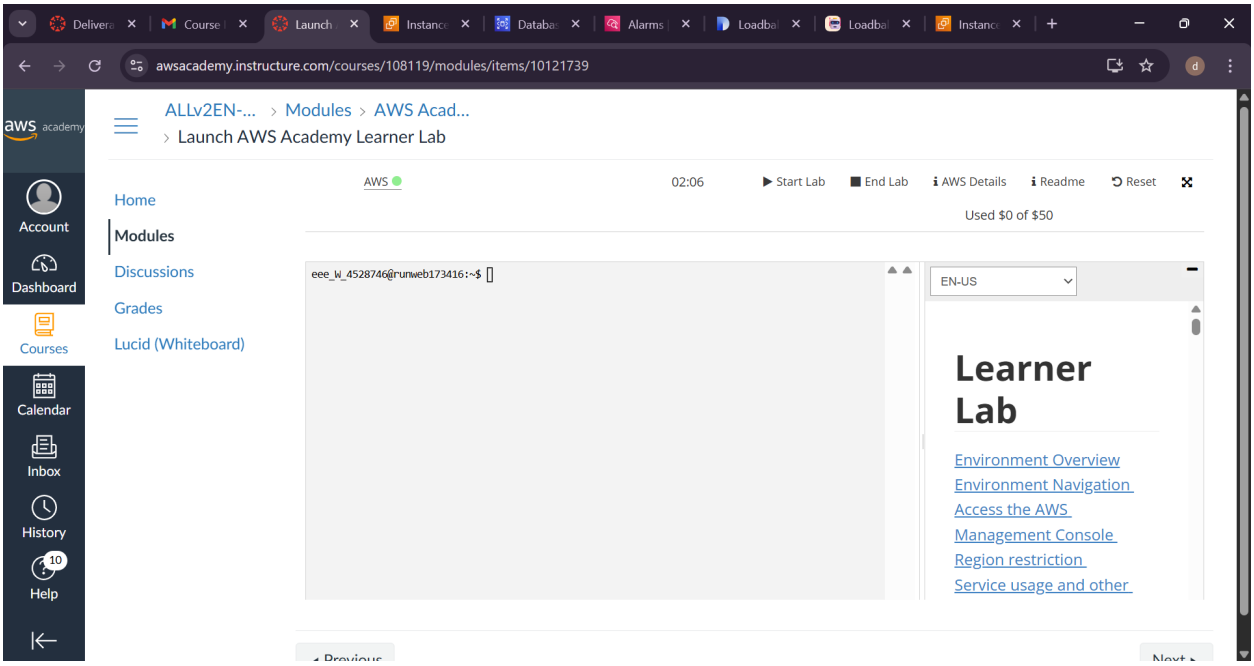
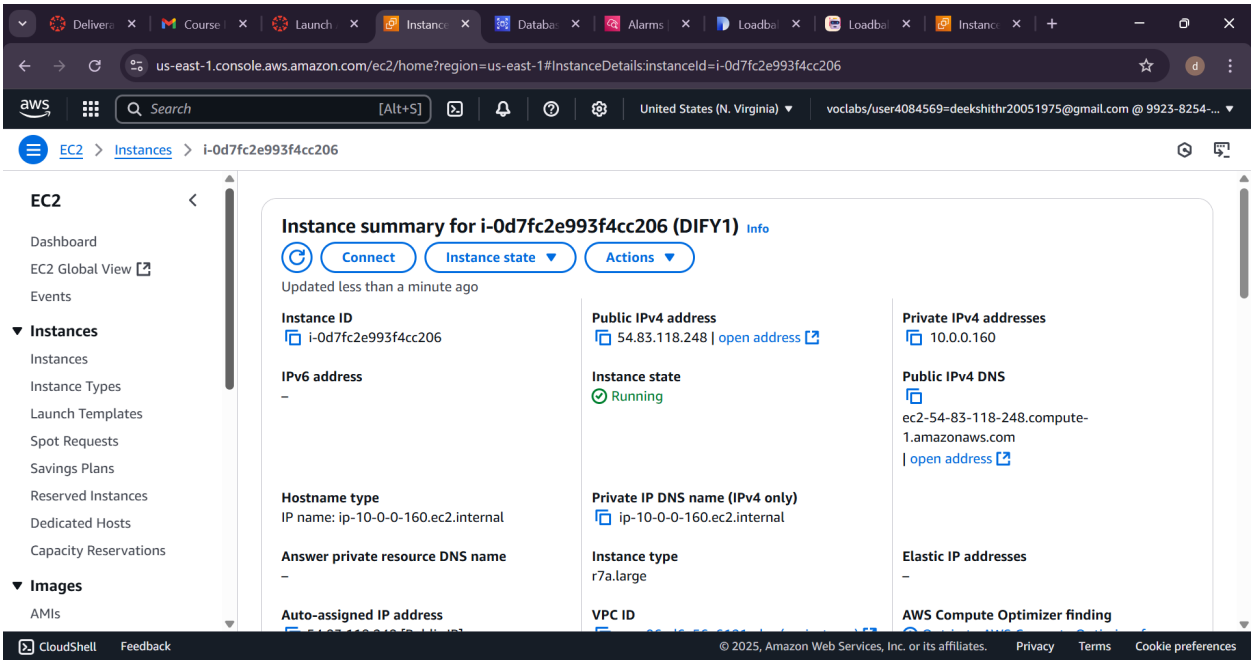
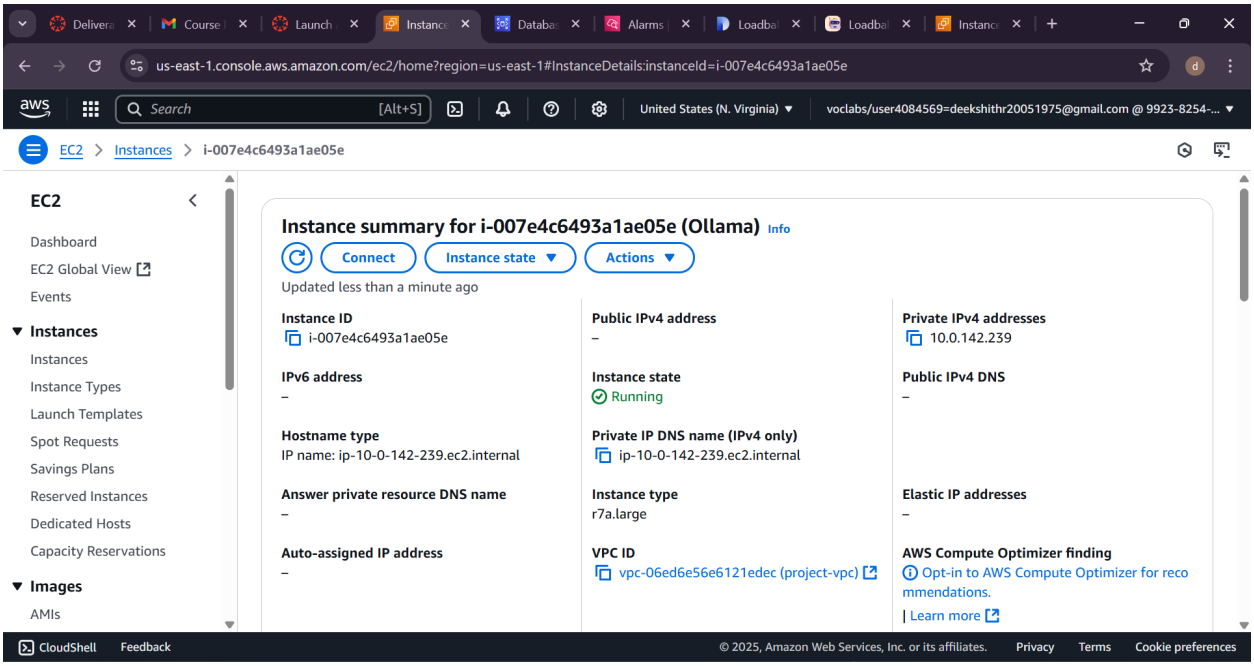
Deliverable #3



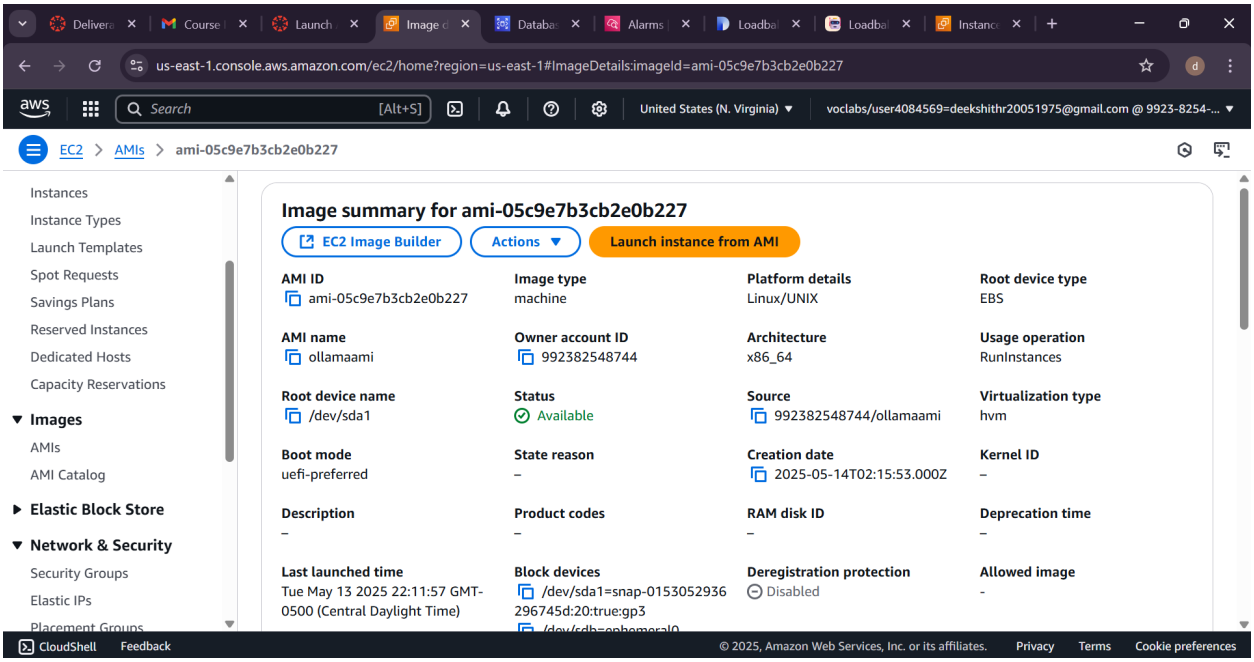
I have Started AWS learners academy lab



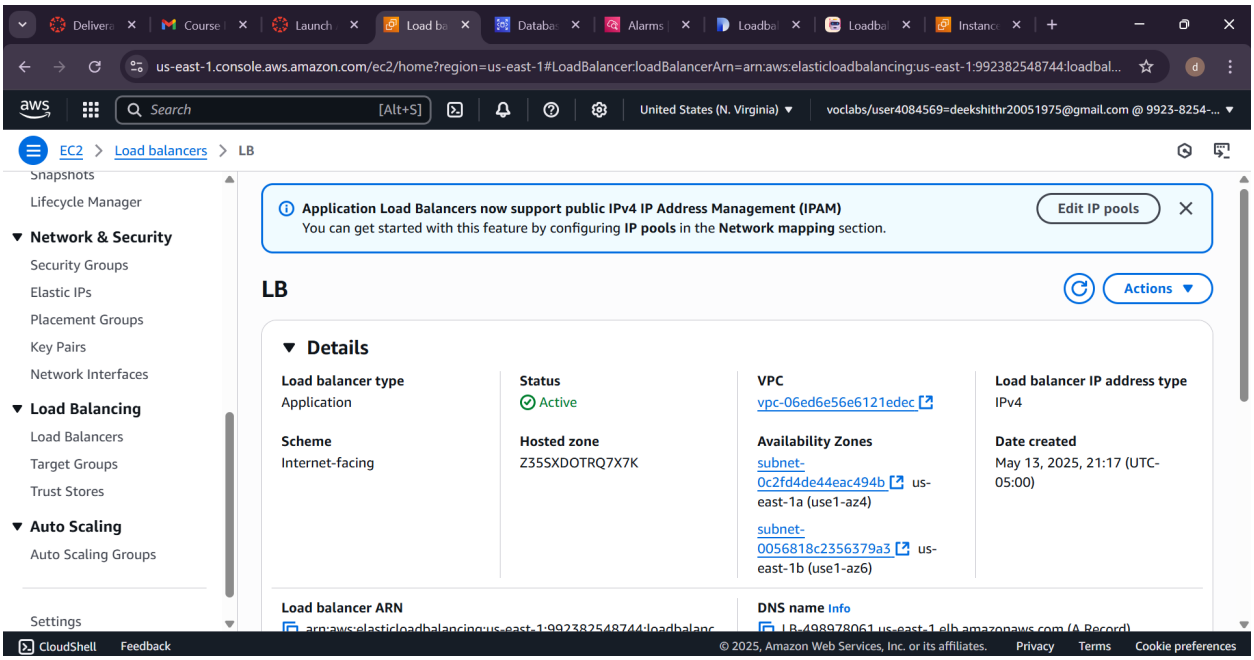
Shows a running EC2 instance (DIFY1) with public IP 54.83.118.248 used to host the Dify frontend. This confirms frontend access to the LLM infrastructure is active and reachable.



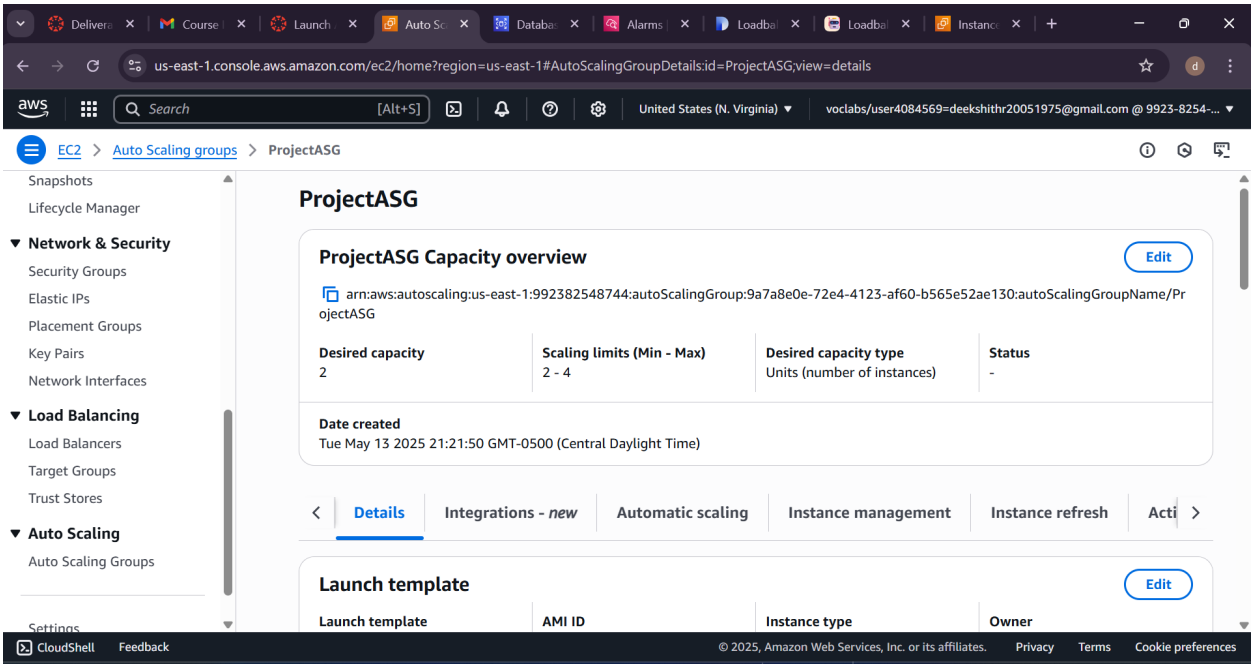
Displays the Ollama LLM server running on EC2 with private IP 10.0.142.239. This instance is part of the load-balanced backend, serving LLM inference requests in a scalable manner.



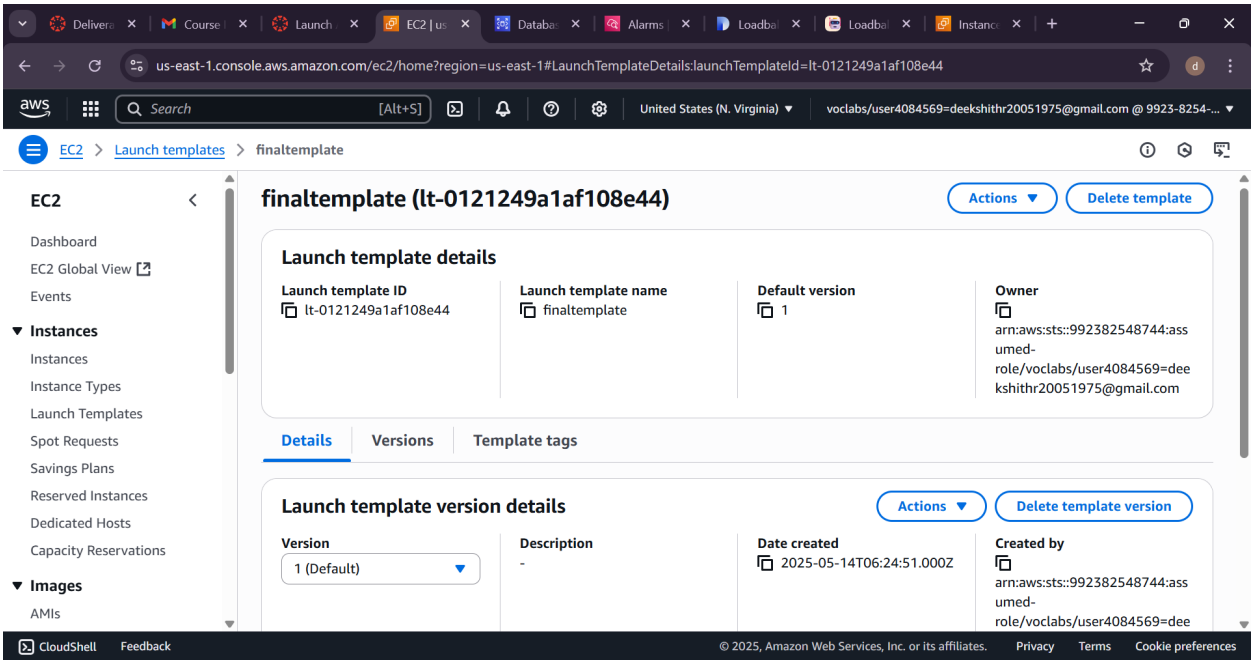
Amazon Machine Image (AMI) named ollamaami, which includes pre installed Ollama and dependencies. This AMI is used in the launch template to ensure consistent and fast provisioning of new EC2 instances for the Auto Scaling Group.



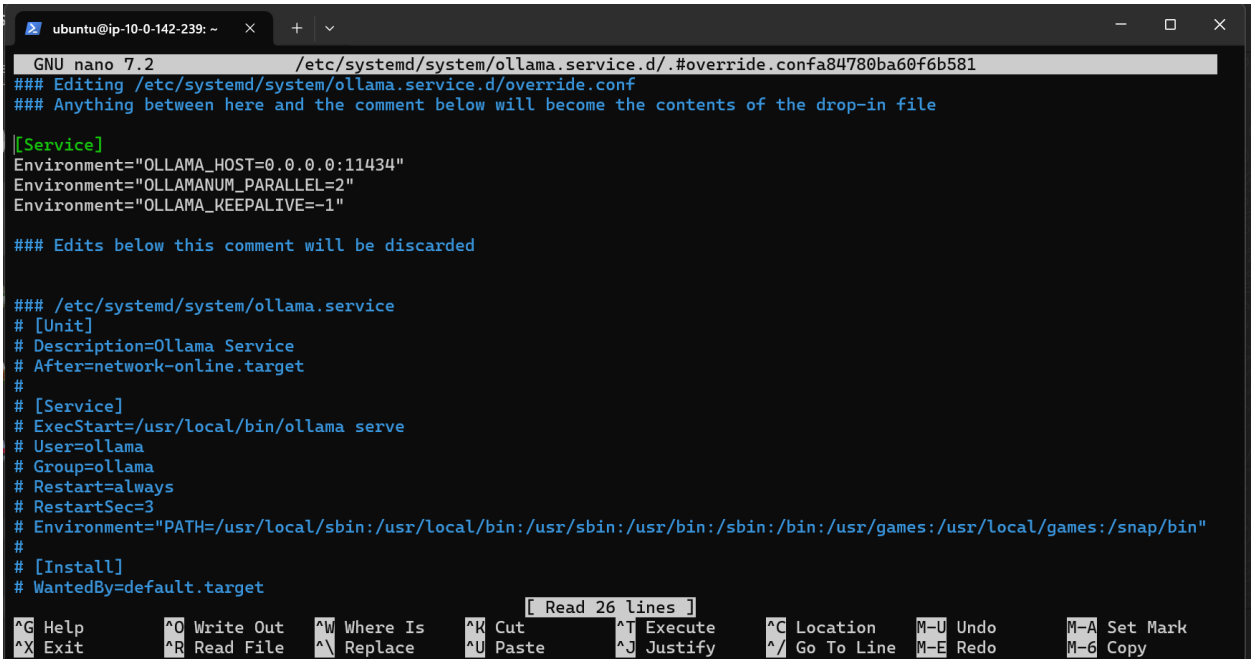
This screenshot shows the Application Load Balancer I configured to route traffic to multiple Ollama EC2 instances. It is internet-facing and currently active, ensuring external access to the LLM backend. The load balancer spans two availability zones for high availability and fault tolerance. This setup improves performance and prevents any single instance from being overloaded.



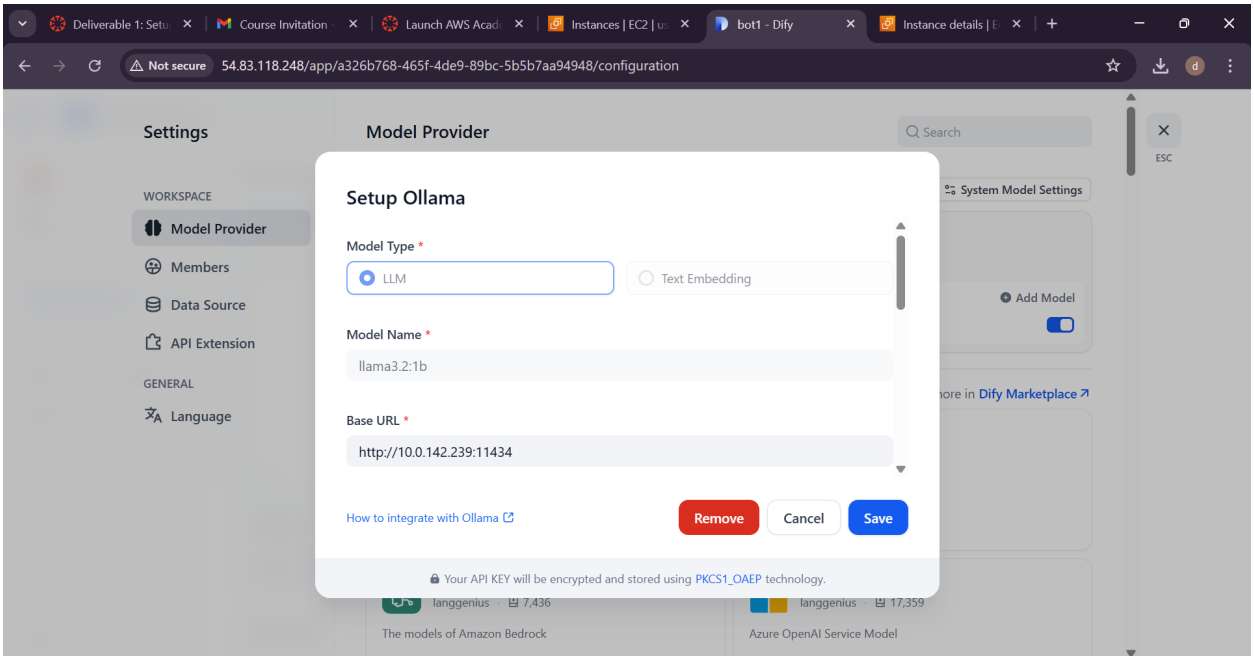
This shows the Auto Scaling Group ProjectASG configured with a desired capacity of 2 instances and scaling limits from 2 to 4. It automatically adds or removes EC2 instances based on demand. This setup ensures consistent performance during high-load inference tasks and supports fault tolerance.



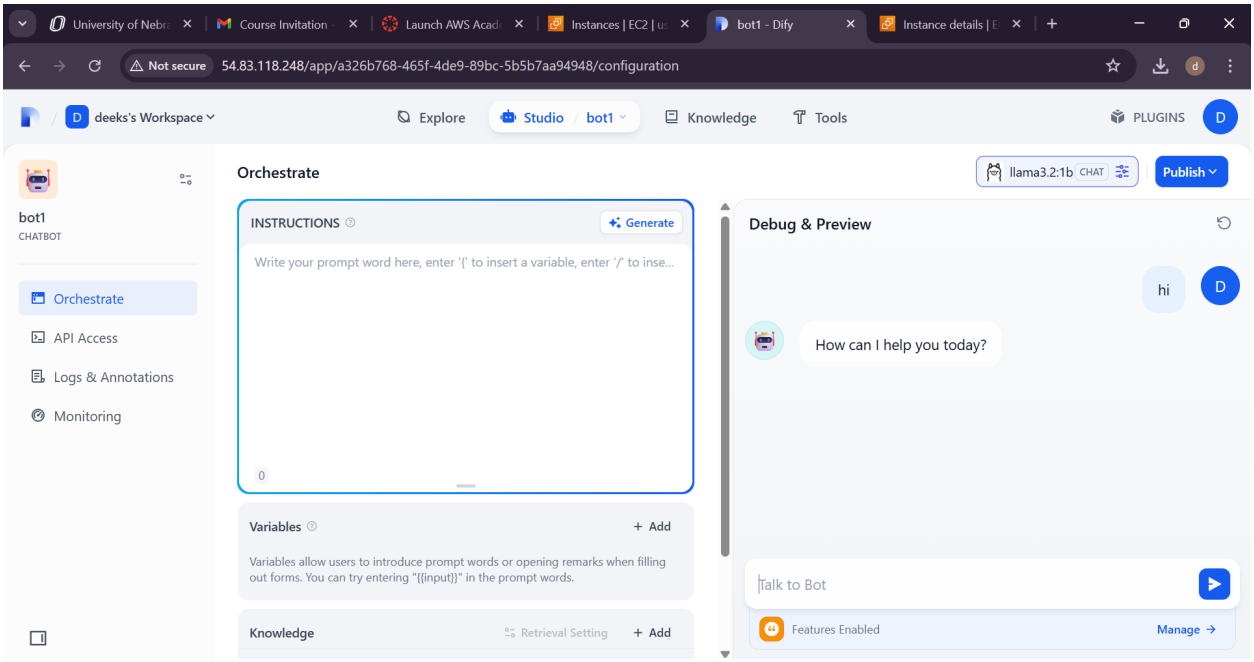
This screenshot shows the launch template named finaltemplate, which is used by the Auto Scaling Group to create new EC2 instances. It includes all the necessary configurations like the AMI, instance type, and startup settings to deploy Ollama servers consistently and quickly when scaling occurs.



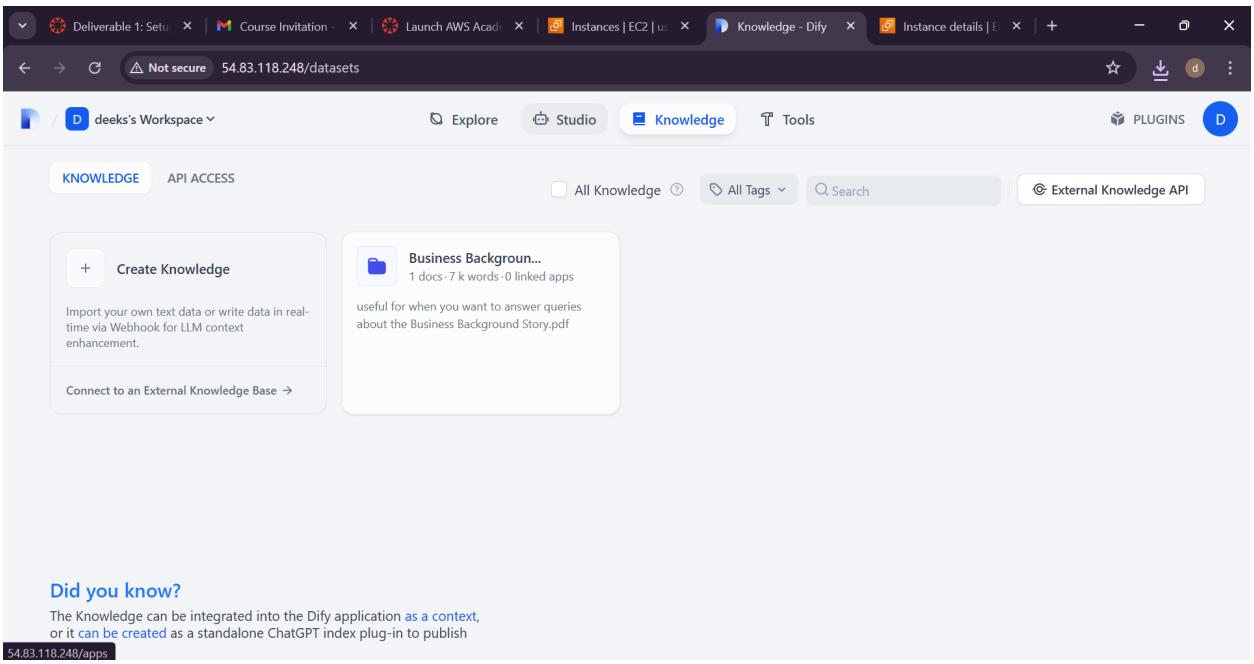
This screenshot shows the Ollama service configuration file on the EC2 instance. Key environment variables like OLLAMA_NUM_PARALLEL=2 and OLLAMA_KEEPALIVE=-1 are set to enable parallel processing and keep the model in memory, improving inference speed and avoiding crashes during high load.



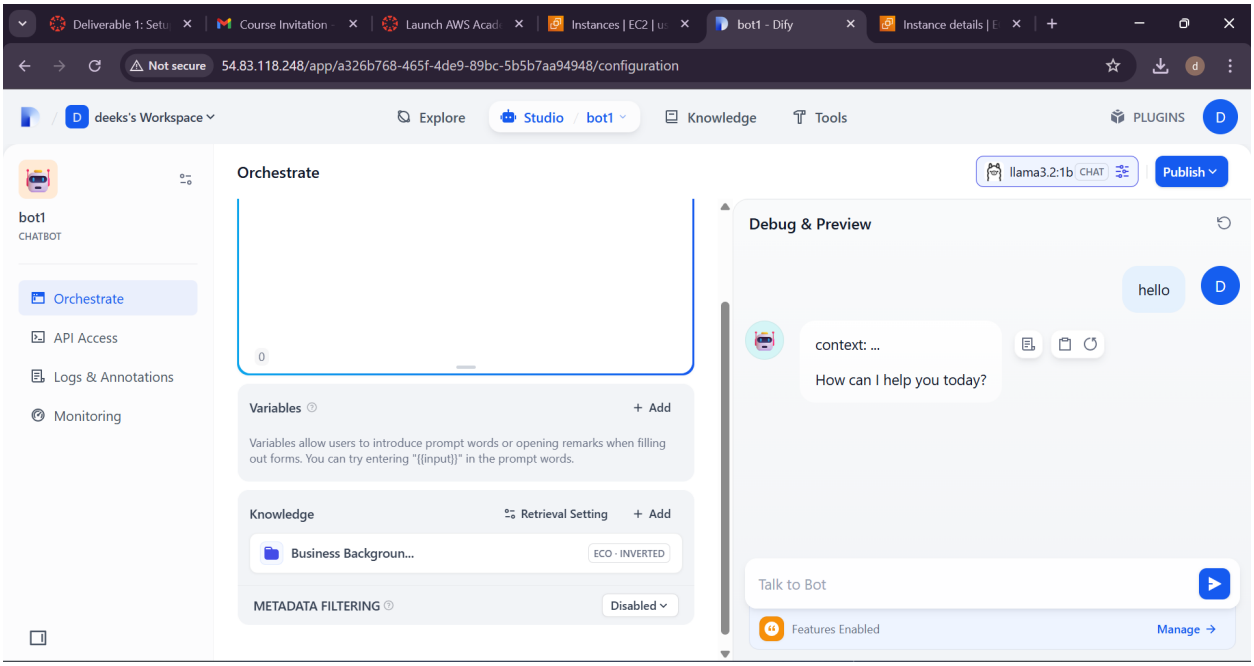
This shows the Dify model provider setup where the Ollama model is connected using its private IP (10.0.142.239:11434). This setup was used before switching to the load balancer, allowing Dify to send LLM queries directly to a single backend instance.



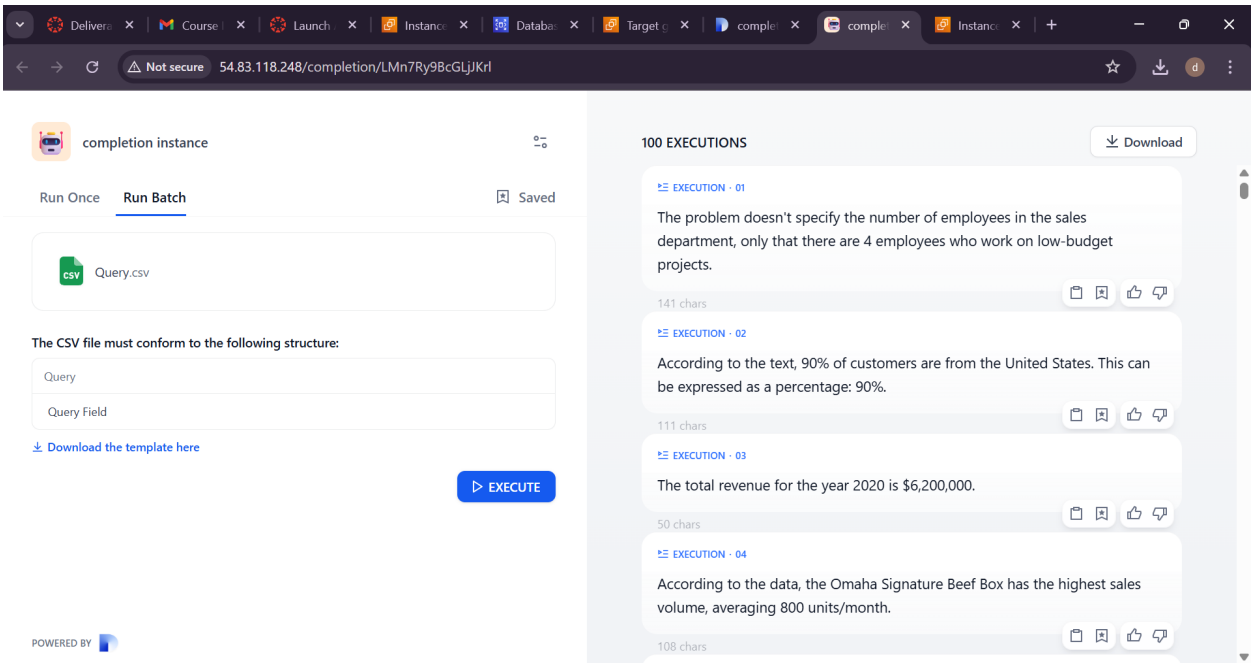
It shows the Dify chatbot interface successfully integrated with the Ollama model. The bot (bot1) responds with “How can I help you today?”, confirming that the backend model is working and ready for user interaction through the UI.



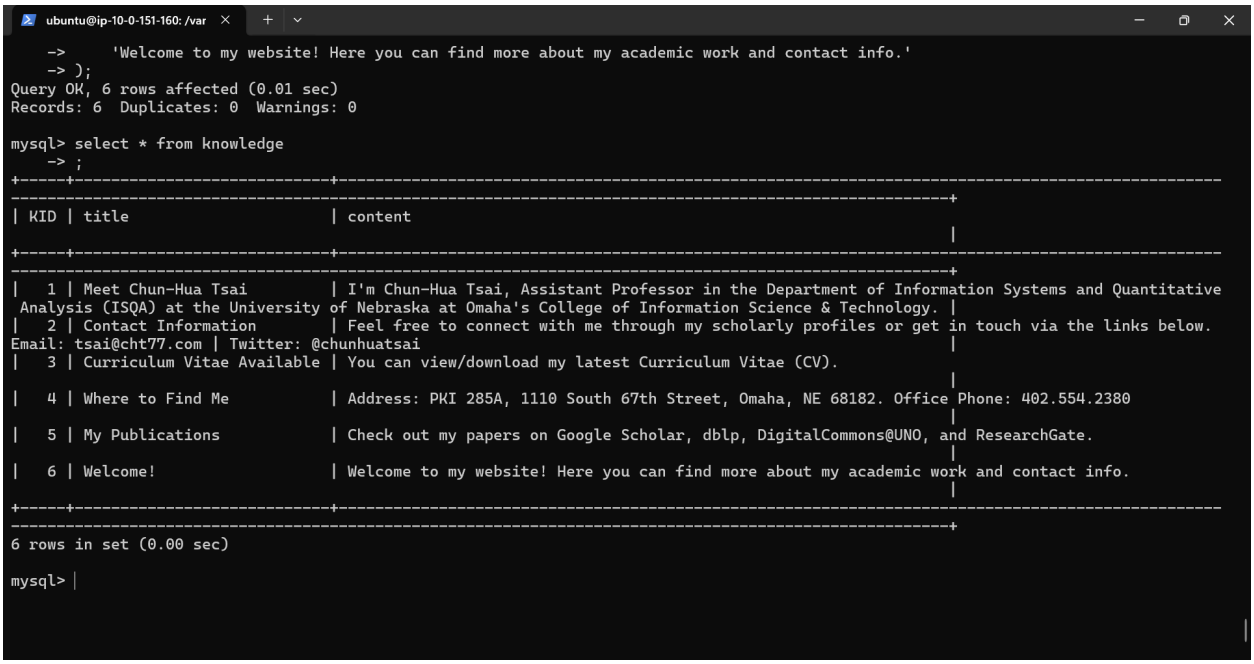
This shows the Knowledge section in Dify, where I added a document titled "Business Background". It’s available for the chatbot to use as context when responding to user queries, enhancing the relevance and accuracy of generated answers.



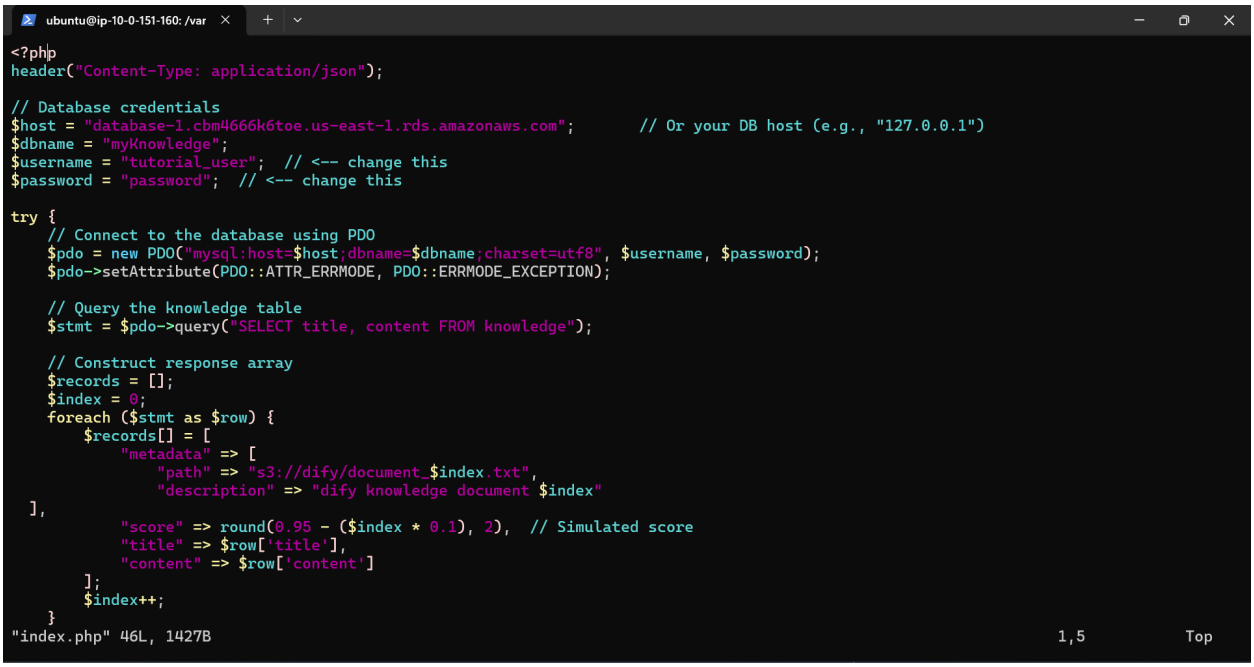
This screenshot shows the chatbot (bot1) using the linked Business Background document as context for responses. The context is activated, and the bot replies with "How can I help you today, confirming successful integration of the knowledge base with the LLM.



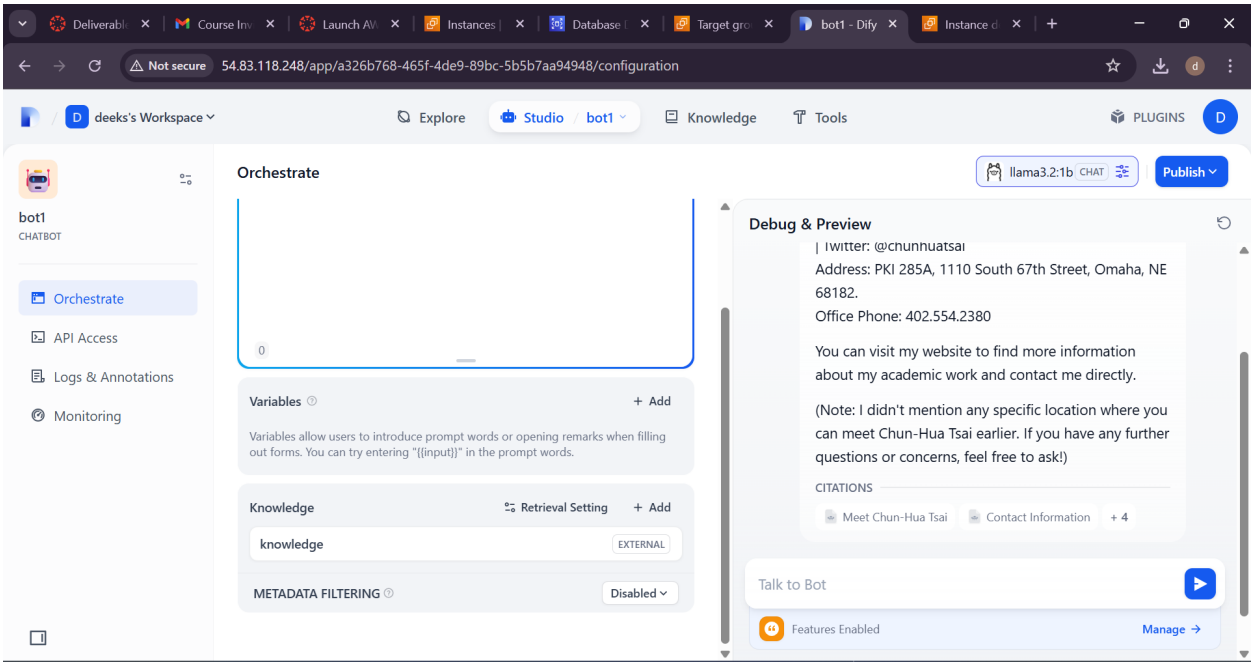
shows the batch completion interface in Dify, where 100 queries from a CSV file were executed. The successful outputs confirm that the load balanced Ollama backend handled large-scale inference efficiently without failures or timeouts.



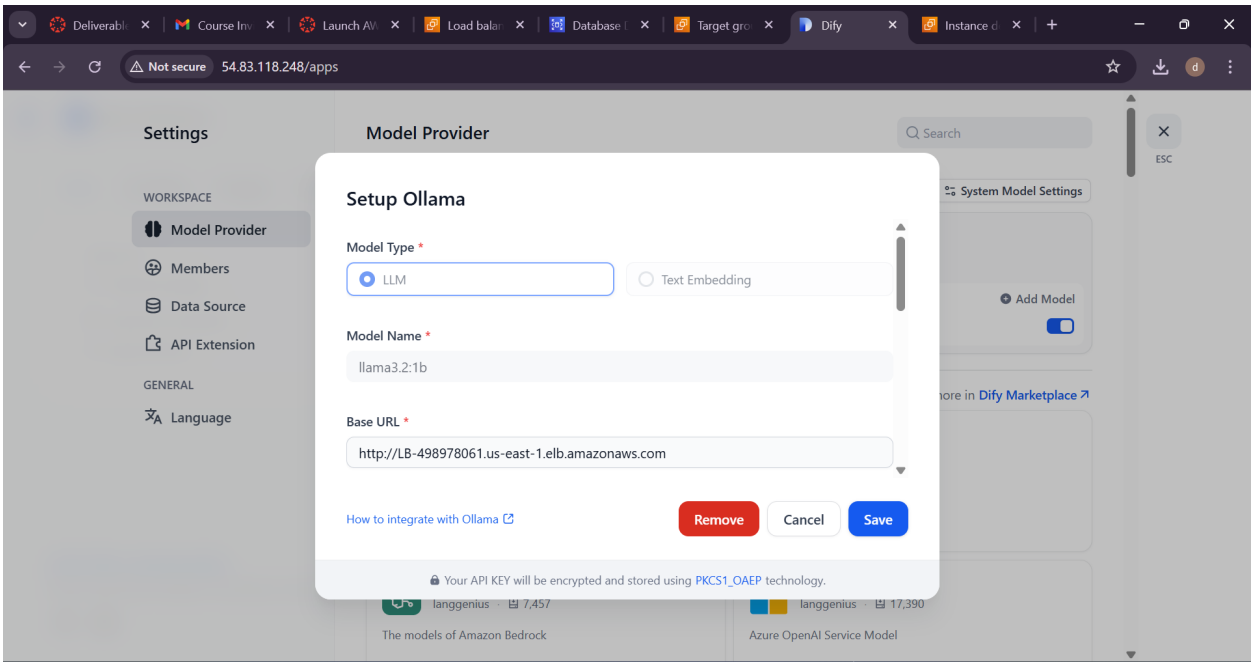
MySQL query result displaying six rows from the knowledge table. These entries serve as structured contextual data that the Dify LLM can use to generate informed responses during chat interactions.



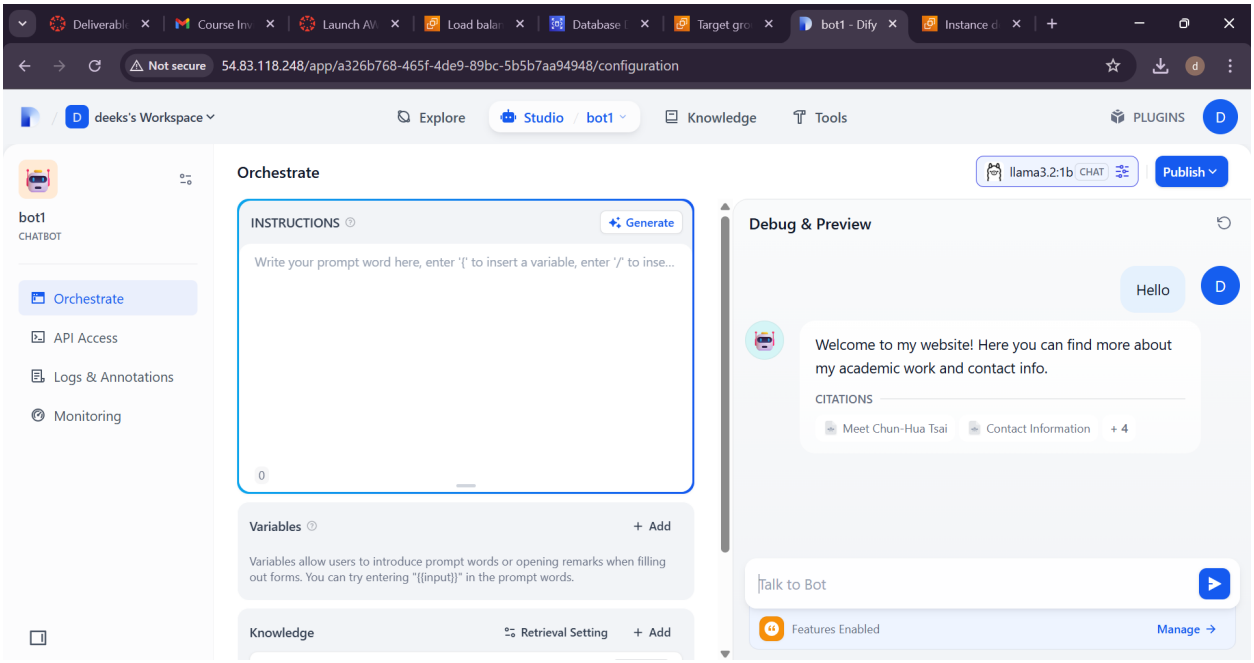
PHP script that connects to an AWS RDS MySQL database and fetches entries from the knowledge table. The results are formatted into a JSON response enabling integration with Dify for dynamic context retrieval during chatbot interactions



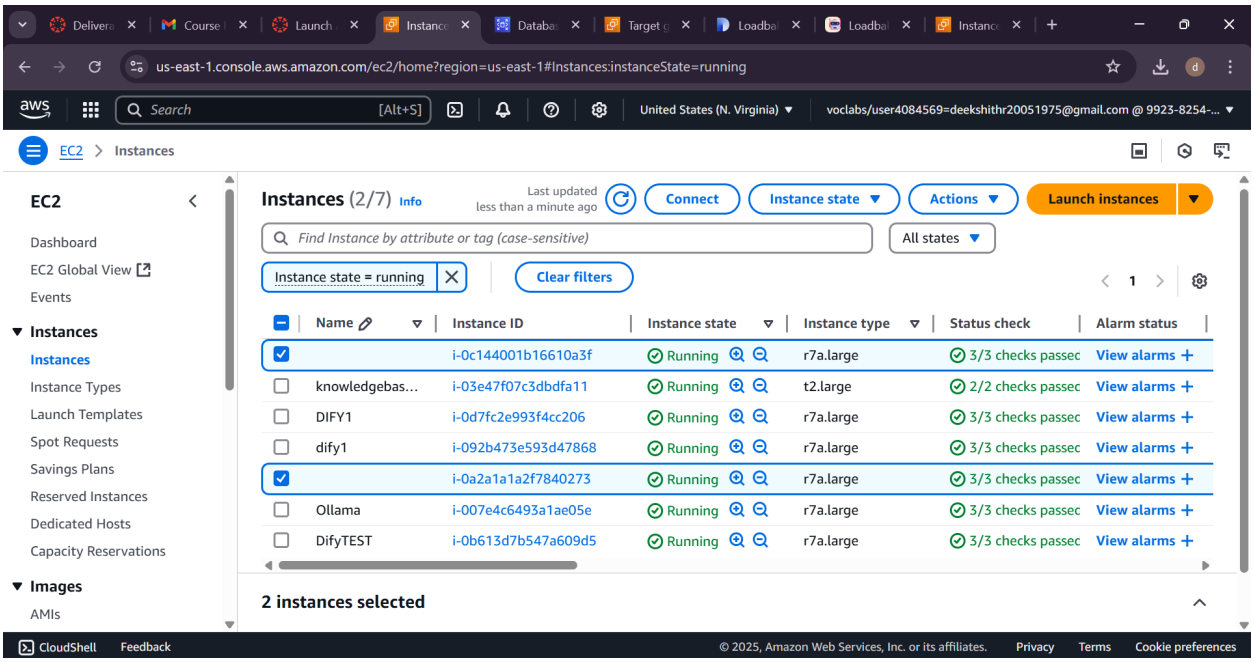
shows the Dify chatbot successfully retrieving and displaying academic contact information from the connected MySQL knowledge base. The response includes detailed data along with source citations, proving that external knowledge integration is working correctly within the LLM pipeline.



he Dify model provider setup updated to use the **Load Balancer DNS URL** instead of a single instance. This change allows Dify to send requests to multiple Ollama servers, improving availability, fault tolerance, and inference speed.

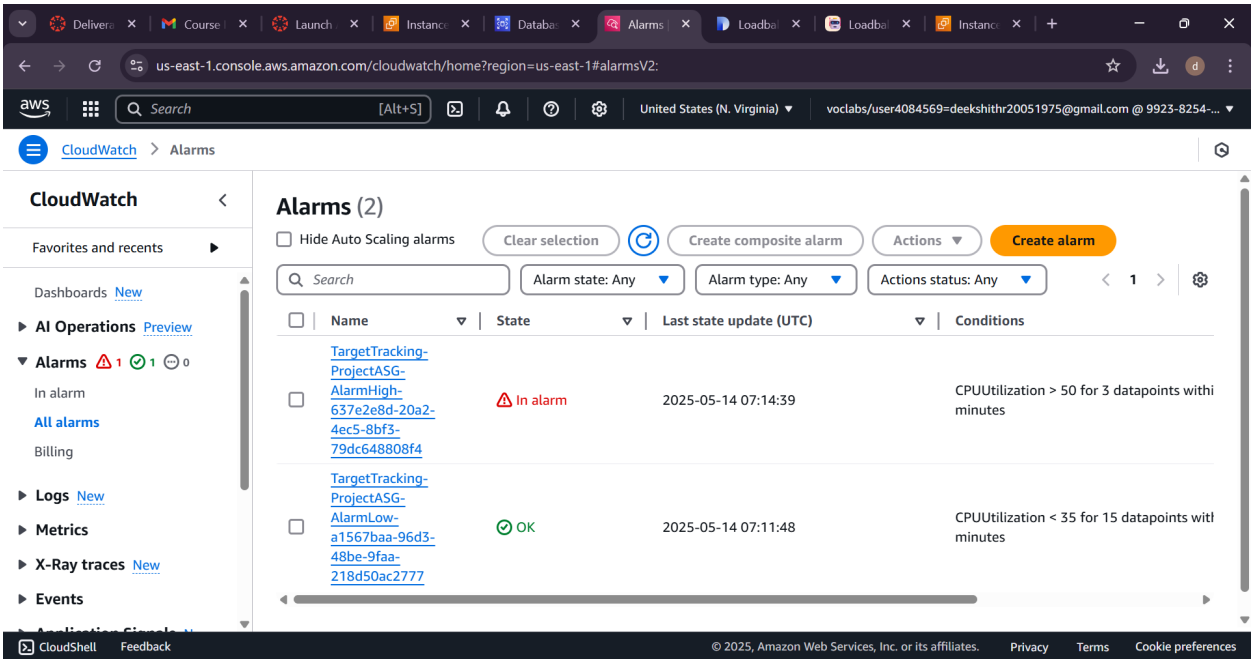


the Dify chatbot successfully responding with information from the knowledge base after the user input "Hello." It confirms that contextual data retrieval and citation features are working correctly with the integrated external knowledge source

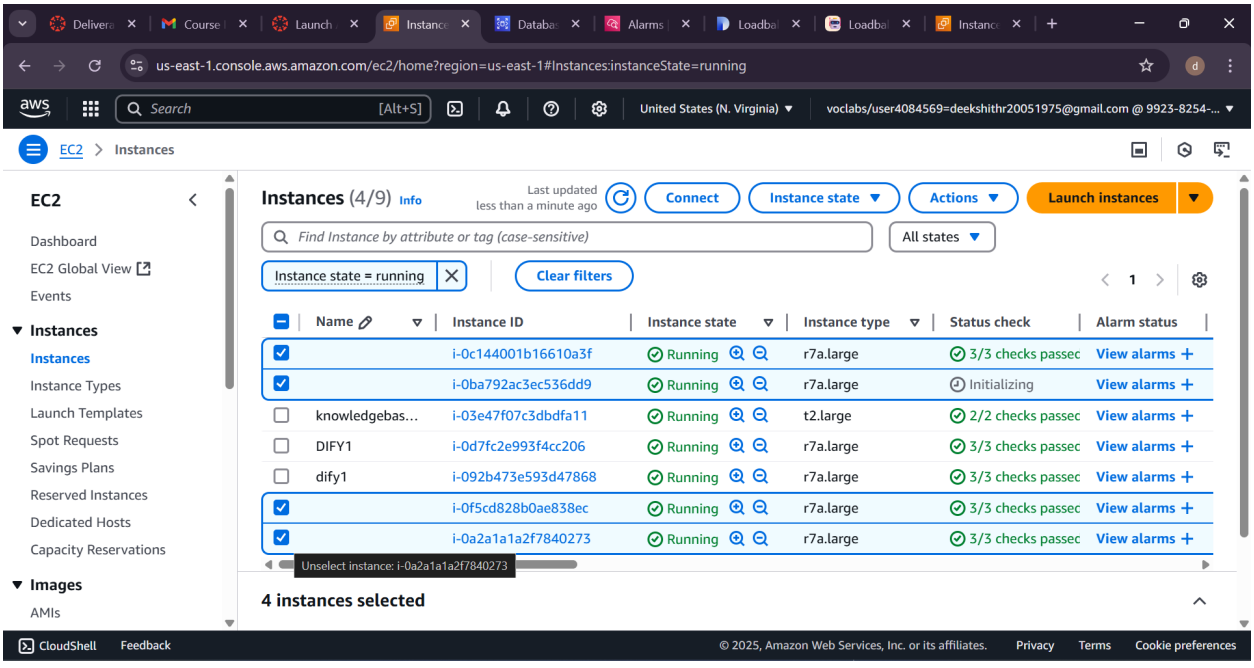


the EC2 dashboard with multiple instances running, including DIFY1, Ollama, knowledgebase, and several autoscaled instances. All have passed status checks,

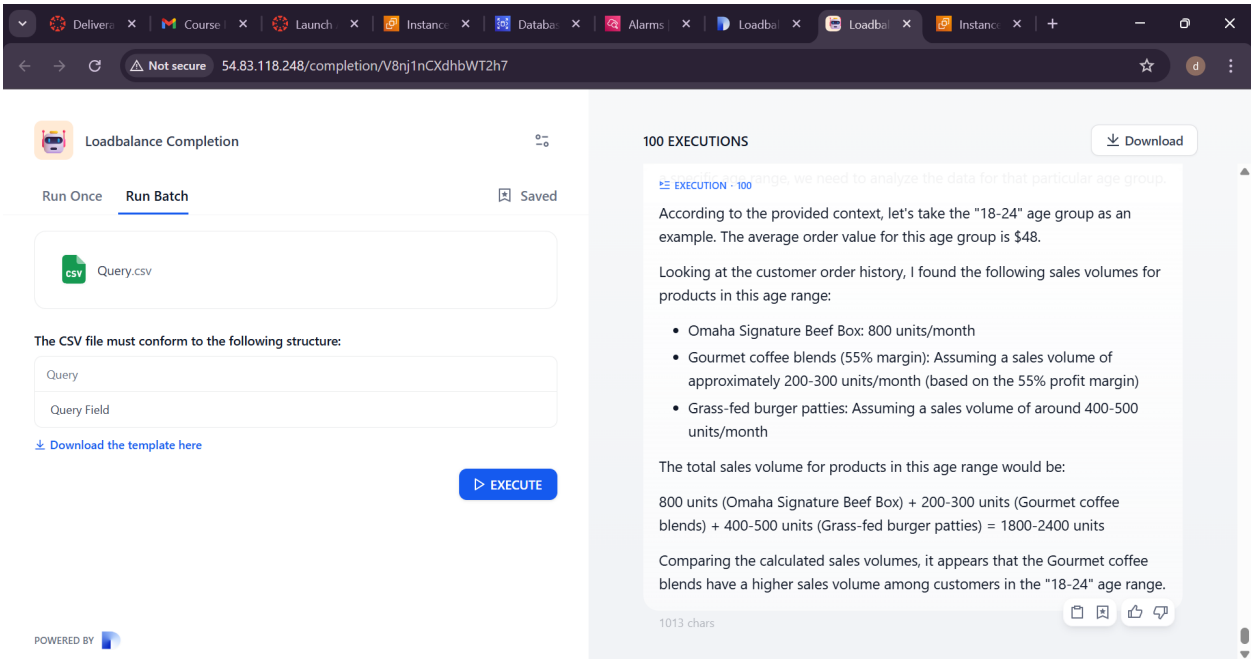
confirming the infrastructure is healthy and ready to handle load-balanced LLM inference requests



This screenshot shows two CloudWatch alarms for the Auto Scaling Group: one is triggered for high CPU usage (>50%), and the other confirms normal CPU load (<35%). This setup ensures instances are automatically scaled based on real-time performance demands.



multiple EC2 instances are actively running, including newly launched ones (Initializing status seen). This indicates that the Auto Scaling Group is functioning correctly by provisioning additional servers in response to system demand.



This screenshot shows the 100th batch execution result in Dify's Loadbalance Completion interface. It confirms that the full batch of inferences was successfully processed, with detailed analysis provided demonstrating improved performance using the load-balanced Ollama backend

1. Load Balancer & Auto-Scaling Setup

I successfully implemented an Application Load Balancer (ALB) in front of multiple Ollama EC2 servers. The load balancer is internet-facing and spans multiple availability zones to ensure high availability. I also created an Auto Scaling Group (ASG) with a scaling range of 2–4 EC2 instances, so the system can automatically adjust based on CPU utilization. This makes the Ollama-based LLM infrastructure highly scalable and fault-tolerant.

2. Switched Dify Model Provider to Load Balancer

To improve reliability and availability, I updated the Dify model provider configuration to point to the load balancers DNS instead of a single private IP. Now, Dify can send inference requests to any healthy Ollama server behind the load balancer, which reduces downtime and speeds up response times under load.

3. Performance Testing with 100 Inference Requests (3 points)

I ran a batch sentiment analysis task using 100 entries from a CSV file. The execution was successful, with all 100 outputs returned without delay or error. Compared to earlier setups where a single instance caused delays or failures, the load-balanced setup processed results much faster and more reliably.
For example, the 100th execution completed with detailed analysis, and system load was distributed efficiently across autoscaled EC2 instances.

4. **Improving LLM Data Analysis Without GPU**

To enhance the performance of data analysis using Large Language Models LLM without relying on GPU servers, I optimized the infrastructure in several ways. First, I configured Ollama with OLLAMA_NUM_PARALLEL=2 and OLLAMA_KEEP_ALIVE=-1, allowing the model to handle multiple requests in parallel while keeping it active in memory to reduce cold-start latency. I also created a custom Amazon Machine Image (AMI) that includes the LLM and all necessary dependencies, enabling new EC2 instances to launch quickly as needed by the Auto Scaling Group. For the knowledge base, I used a MySQL database, which can be further improved by migrating to Aurora Serverless for better scalability or adding read replicas to handle more queries simultaneously. Additionally, caching frequently accessed knowledge entries with tools like Redis or Memcached would significantly reduce database load and speed up responses. Lastly, by offloading static or less intensive tasks to lightweight EC2 instances or Lambda functions, the system can maintain responsiveness even under high demand all without the need for GPU acceleration.

Feedback AI Agent :Using the AI agent through Dify connected to Ollama was a great experience. It was easy to set up and provided a clean interface for both testing and customizing chatbot behavior. Once I integrated the external knowledge base and switched to the load-balanced backend, the responses became more accurate and consistent. I especially liked how the batch inference feature let me test 100 entries quickly, which helped me evaluate the systems performance under load. Overall, the AI agent felt responsive and reliable, and it gave me a good understanding of how LLMs can be deployed in real-world applications

Difficulties Faced: So I was Really hard to understand that the Ollama it was Updated when I tried to connect to ollama It was not connecting One of the main difficulties I faced was during the initial setup of the Ollama service, especially making sure the model stayed active and didn’t crash under load. I had to fine-tune environment variables like OLLAMA_NUM_PARALLEL and OLLAMA_KEEP_ALIVE to avoid issues. Another challenge was ensuring the health checks for the load balancer were properly configured—by default, the LLM response time was too slow and caused instances to fail health checks. I also needed to SSH into new EC2 instances launched by the Auto

Scaling Group to verify if the model was running correctly, which added extra steps during testing. Finally, integrating the MySQL knowledge base with Dify required careful handling of PHP and database credentials to ensure secure and smooth data flow.